Propat-python Kinematics Guide

Contents

DCM

Rotation matrix

Euler angle

Rodrigues rotation

Quaternion

Euler Kinematics

Quaternion Kinematics

**<1>** **Direction Cosine Matrix (DCM)**

**DCM** is a method to analyze a relationship between the coordinates(frames).

Mathematically, we could reduce DCM from one-axis rotation.

(Note that “right axis rule” is applied in this document.)

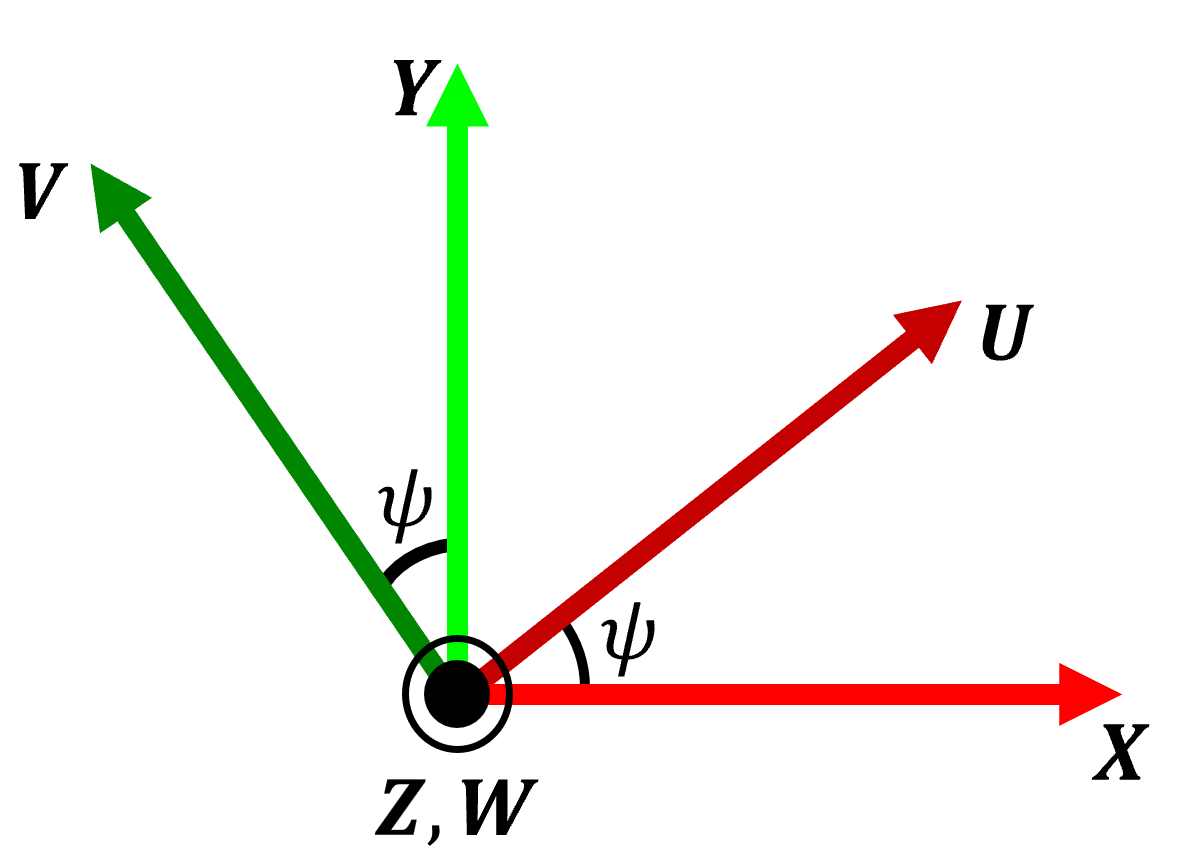


Figure 1. Explanation for "DCM" by z-axis rotation

Let’s define two frames as XYZ frame and UVW frame respectively. Assuming that these frames are composed of orthogonal unit vectors ( , ), then we could write their relationship in matrix notation as follows:

(1)

where is called the ***direction cosine***; the cosine of the angle between and .

For Simplicity, we write Eq.(1) as follows:

(2)

where is a DCM to frame from frame and is a vector notation.

DCM is also an orthonormal matrix. Thus, we have

(3)

which means that

(4)

Using Eq.(1), we can find that DCM for z-rotation( for z-rotation angle) is as below:

(5)

By brevity, we often skip frame notation.

Similarly, for x-rotation and y-rotation, we can apply these calculation process. Then, we could get DCM for each axis rotation( for x-rotation angle and for y-rotation angle) as follows:

(6)

In propat-python, DCM is implemented as below:

|  |  |
| --- | --- |
| x-rotation | rotmax(angle) |
| y-rotation | rotmay(angle) |
| z-rotation | rotmaz(angle) |

Table 1. DCM function list

The function names could make confusion to users, but we follow original PROPAT functions’ name.

**<2> Rotation Matrix (RM)**

RM is usually called as DCM. But for exact classification, we define RM as a different rotation method.

Mathematically, we could reduce RM as one-axis rotation.

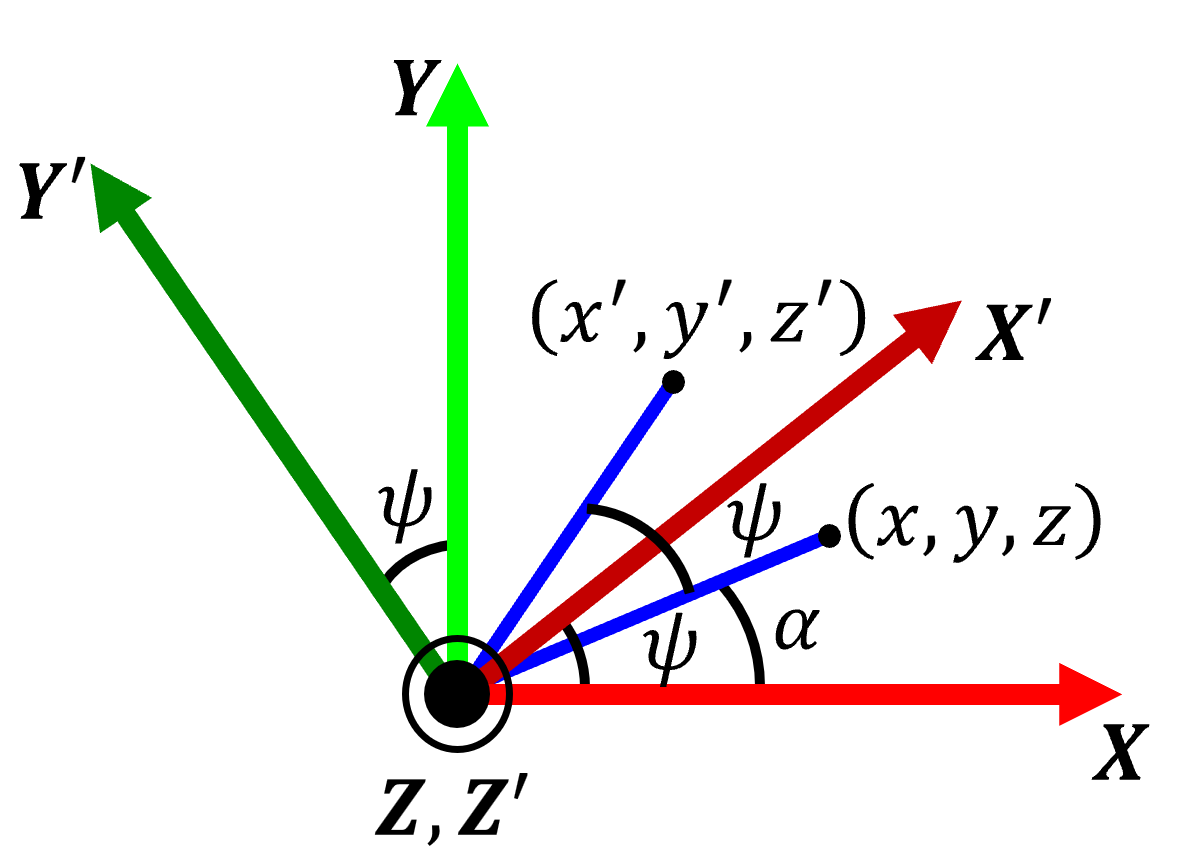


Figure 2. Explanation for “RM” by z-rotation

Let’s define two frames as XYZ frame and X’Y’Z’ frame respectively and select arbitrary point in XYZ frame. Then, we could write their relationship in matrix notation as follows:

(7)

(8-1)

(8-2)

Using Eq.(7)-(8), we can find that RM for z-rotation( for z-rotation angle) is as below:

(9)

Therefore, we could reduce RM for z-rotation as follows:

(10)

Similarly, for x-rotation and y-rotation, we can apply these calculation process. Then, we could get RM for each axis rotation( for x-rotation angle and for y-rotation angle) as follows:

(11)

(12)

(13)

In propat-python, RM is implemented as below:

|  |  |
| --- | --- |
| x-rotation | rotmax\_rx(angle) |
| y-rotation | rotmay\_ry(angle) |
| z-rotation | rotmaz\_rz(angle) |

Table 2. RM function list

As you can see, RM is a transpose of DCM.

(14)

There are several fields that use RM as a “golden rule” for analyzing their fields. That is why many people get in trouble with selecting which one is more suitable for their purpose.

Difference between RM and DCM arises from perspective of interpreting rotation. To explain it, let’s define arbitrary 3 frames: A, B and C.

As a perspective of RM, Rotation is analyzed from beginning. That is,

DCM, on the other hand, Rotation is analyzed from end . That is,

Shortly and intuitively saying, one who wants to analyze frames’ relationship with ***“forward propagation”*** then RM is suitable. With the same way, one who wants to analyze frames’ relationship with ***“back propagation”*** then DCM is suitable.

In this project, we use **“DCM”** method because we need ***“back propagation”*** rotation analysis with respect to ECI frame; That is, ECI frame is a standard frame to observe other frames like LVLH, Body frame etc.(e.g. Body→LVLH→ECI)

**<3> Euler angles**

Euler angles are set of three angles that define the orientation of an object in three-dimensional space. There are 12 types of rotation sequences that can be represented by Euler angles. The table below shows all types of rotation sequences using Euler angles.

|  |  |  |
| --- | --- | --- |
| X-Y-Z | Y-Z-X | Z-X-Y |
| X-Z-Y | Y-X-Z | Z-Y-X |
| X-Y-X | X-Z-X | Y-X-Y |
| Y-Z-Y | Z-X-Z | Z-Y-Z |

Table 3. Types of rotation sequences

In propat-python, there are 4 Euler rotation sequences like the table below:

|  |  |
| --- | --- |
| X-Y-Z | exyzrmx(euler\_angles) |
| Z-X-Y | ezxyrmx(euler\_angles) |
| Z-X-Z | ezxzrmx(euler\_angles) |
| Z-Y-X | ezyxrmx(euler\_angles) |

Table 4. Euler angle rotation function list

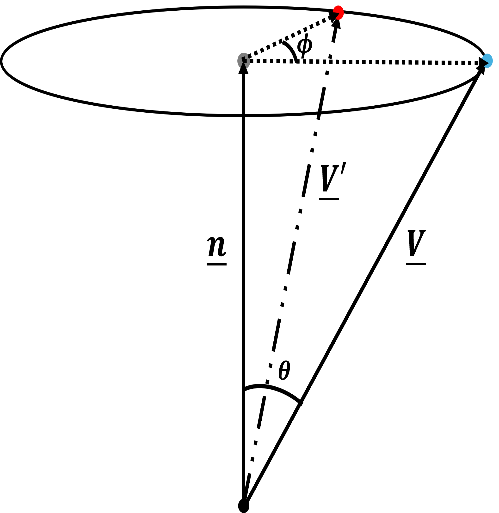
In similar way, it is possible to get Euler rotation sequence from rotation matrix by analyzing elements of rotation matrix.

In propat-python, there are 4 angle abstraction functions like the following table:

|  |  |
| --- | --- |
| X-Y-Z | rmxexyz(rot\_mat) |
| Z-X-Y | rmxezxy(rot\_mat) |
| Z-X-Z | rmxezxz(rot\_mat) |
| Z-Y-X | rmxezyx(rot\_mat) |

Table 5. Euler angle abstraction function list

**<4> Rodrigues Rotation**

블랙, 어둠이(가) 표시된 사진

자동 생성된 설명블랙, 어둠이(가) 표시된 사진

자동 생성된 설명

Figure 3. Explanation for Rodrigues rotation

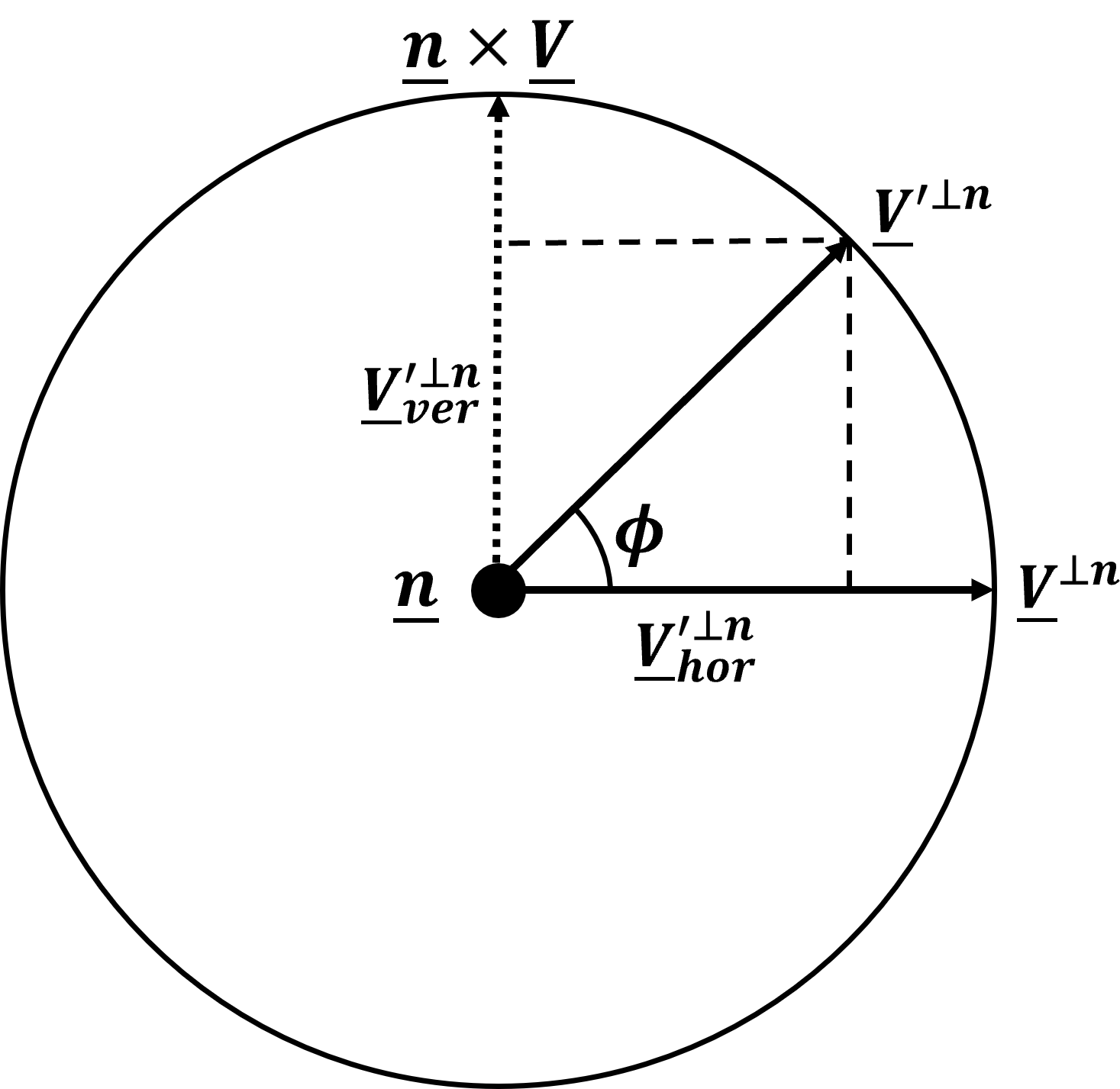


Figure 4. Upper view of Rodrigues rotation

Rodrigues rotation is a method used in three-dimensional geometry to rotate a vector by a specified angle about an axis defined by a unit vector. Let’s assume that there are ‘n’ unit vector and ‘v’ vector like Figure 3. Considering ‘n’ vector as a central axis and ‘v’ vector as a rotating vector around the central axis, Figure 4. could be drawn clearly.

From Figure 3., we could find that

and (15)

From Figure 4., we could also find that

(16)

(17)

(18)

Using Eq.(15)-(18),then ‘ v’ ’ vector can be expressed in terms of the relationship between ‘n’ vector and ‘v’ vector.

(19)

For reduction, we could represent Eq.(15) as below:

and (20)

Then Eq.(19) could be derived as follows:

(21)

Note that cross product can be represented as matrix multiplication in the following manner.

(22)

Then Eq.(21) is reformulated as follows:

(21)

In propat-python, there is Rodrigues rotation function.

|  |  |
| --- | --- |
| Rodrigues rotation | eulerrmx(euler\_angle, euler\_vector) |

Table 6. Rodrigues rotation function

Given Rotation matrix , is equivalent to the matrix below:

(22)

From (22), central axis(‘n’ vector) and rotation angle(‘θ’) can be found:

(23)

(24)

If rotation angle is zero, then central vector is [1,0,0] and rotation angle is 0.

If rotation angle is 180 degree, then the trace of the rotation matrix will be ‘-1’ because cos(π)=-1 and diagonal elements and off-diagonal elements of the rotation matrix will be

(25)

(26)

Considering (25)-(26), central vector can be determined.

In propat-python, there is conversion function for abstracting Euler angle and central vector from rotation matrix.

|  |  |
| --- | --- |
| Rodrigues vector & angle abstraction | rmxeuler(rot\_mat) |

Table 7. Rodrigues vector & angle abstraction function

**<5> Quaternion**

Quaternion is a method for representing the orientation of objects in kinematics. To fully understand its use, it's important to explain why quaternions are utilized in kinematics and why it uses a 4-dimensional vector to avoid gimbal lock. However, these details are not the main focus of this documentation. Instead, the definition and properties of quaternion will be described.

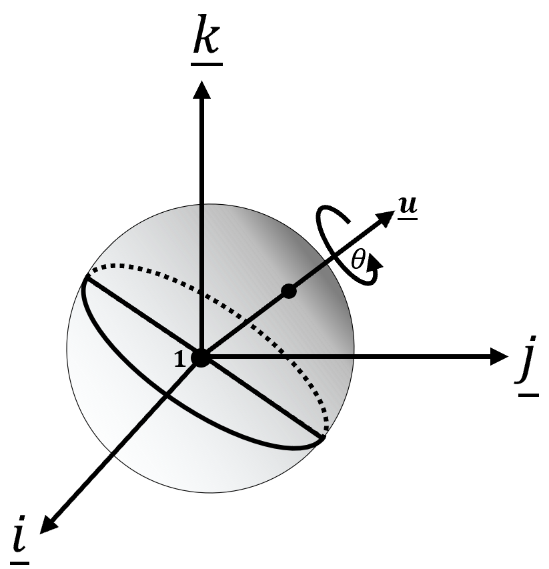


Figure 5. Quaternion

Definition of Quaternion is below:

(27)

And properties of quaternion vector part are below:

(28)

(29)

(30)

(31)

(32)

Shortly, quaternion is **“non-commutable”**.

Properties of quaternion are below:

(33)

(34)

(35)

(36)

(37)

(38)

(39)

(40)

(41)

(42)

(43)

Normally, unit quaternion is used for representing rotation.

Figure 5. shows quaternion rotation.

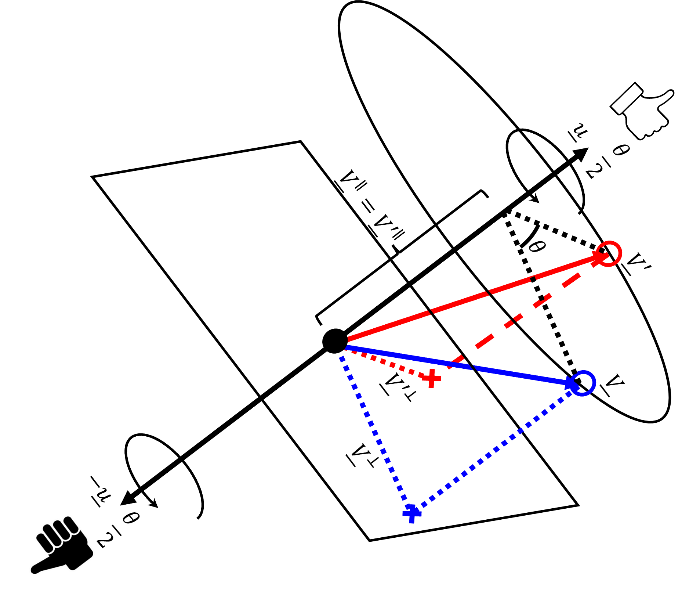


Figure 6. Quaternion rotation

Quaternion represents rotation in three-dimensional space in a way that may be interpreted as involving complex, multi-axis interactions that may be seen as “dual-axis rotation” or “right & left-hand rotation”. For using quaternion as a rotation representation, this analysis is effective despite being mathematically incorrect. This interpretation helps in practical implementation and understanding of quaternion rotation formulas.

If a quaternion is a unit quaternion, real part of it lies between -1 and 1. Additionally, by Euler’s formula, rotation in complex space can be represented as follows:

(44)

By Eq.(44), “V’” can be rewritten like this:

(45)

Note that these theorems can be derived from Eq.(28)-(45).

(46)

(47)

Using Eq.(46) and Eq.(47), Eq.(45) can be reformulated as follows:

(48)

If quaternion is a unit quaternion, then Eq.(48) can be rewritten as below:

(49)

If Eq.(48) is transformed into matrix behavior,

(50)

From Eq.(49), quaternion elements can be found as follows:

(51)

(52)

(53)

(54)

For Z-axis rotation(Yaw), quaternion can be expressed as below:

(55)

In same way, rest of axis rotations can be analyzed as below:

(56)

(57)

Using (55)-(58),ZYX-rotation as an example, rotation matrix can be derived as follows:

(58)

In propat-python, there are several quaternion-related functions like table below:

|  |  |
| --- | --- |
| Rotation to quaternion | rmxquat(rot\_mat) |
| Quaternion multiplication  (Quaternion matrix) | quat\_matrix(q) |
| Quaternion inversion | quat\_inv(q) |
| Quaternion normalization | quat\_norm(q) |
| Quaternion production | quat\_prod(quat1,quat2) |
| Quaternion Unitization | quat\_unity(q) |
| XYZ to Quaternion | exyzquat(euler\_angles) |
| ZXZ to Quaternion | ezxzquat(euler\_angles) |
| Quaternion to XYZ | quatexyz(euler\_angles) |
| Quaternion to ZYZ | quatexyz(euler\_angles) |

Table 8. quaternion function list